Circular Coinduction in Coq Using Bisimulation-Up-To Techniques*

Jörg Endrullis[1] Dimitri Hendriks[1] Martin Bodin[2]

[1] VU University Amsterdam, Department of Computer Science
[2] INRIA Rennes & ENS Lyon

j.endrullis@vu.nl r.d.a.hendriks@vu.nl martin.bodin@inria.fr

Abstract. We investigate methods for proving equality of infinite objects using circular coinduction, a combination of coinduction with term rewriting, in the Coq proof assistant. In order to ensure productivity, Coq requires the corecursive construction of infinite objects to be guarded. However, guardedness forms a severe confinement for defining infinite objects, and this includes coinductive proof terms. In particular, circular coinduction is troublesome in Coq, since rewriting usually obstructs guardedness. Typically, applications of transitivity are in between the guard and the coinduction hypothesis. Other problems concern the use of lemmas, and rewriting under causal contexts. We show that the method of bisimulation-up-to allows for an elegant rendering of circular coinduction, and we use this to overcome the troubles with guardedness.

1 Introduction

As any construction of infinite objects, constructive bisimilarity proofs have to be *productive*. That is, it has to be guaranteed that the proof term has an infinite constructor normal form with respect to the lazy evaluation of the calculus at hand [5]. One way of ensuring productivity is by *guarded corecursion* [5,11]. Guardedness is a simple syntactic criterion implemented in proof assistants based on type theory like Coq [4] and Agda [2]. A corecursive definition is guarded if every corecursive call is guarded by at least one constructor of the coinductive type we are building a term in, and only by such constructors. Then, in the infinite process of unfolding a guarded definition, evermore building blocks of the infinite structure are produced, yielding in the limit a term consisting of constructors only.

Guardedness can be easily checked and it is readily seen why guarded corecursion implies productivity. On the other hand, guardedness is notorious for confining the programmer to a restricted set of tools for defining coinductive objects. Already the most simple examples of productive definitions fail to be guarded. Coquand [5] considers the following corecursive definition

$$\mathsf{nats} = 0 :: \mathsf{map}\ (\lambda n.\ n+1)\ \mathsf{nats} \qquad \mathsf{map}\ f\ (x :: s) = f\ x :: \mathsf{map}\ f s \tag{1}$$

* This research has been funded by the Netherlands Organization for Scientific Research (NWO) under grant numbers 639.021.020 and 612.000.934.

of the sequence of natural numbers $\mathsf{nats} = 0::1::2::\ldots$; where $::$ is the constructor of the coinductive type of infinite sequences, or streams as we call them. The definition of nats is clearly productive, yet it is not guarded, for the recursive call is argument of the map function.

A similar problem occurs for definitions of morphic sequences (see Section 2), like the following definition of the Thue-Morse sequence $\mathsf{M} = 0::1::1::0::1::0::0::1::1::0::0::1::0::1::1::0::\ldots$,

$$\mathsf{M} = 0 :: \mathsf{tail}\ (\mathsf{h}\ \mathsf{M}) \qquad \mathsf{h}\ (0 :: s) = 0 :: 1 :: \mathsf{h}\ s \qquad \mathsf{h}\ (1 :: s) = 1 :: 0 :: \mathsf{h}\ s \tag{2}$$

where $\mathsf{tail}\ (x :: s) = s$. The corecursive call of M is not a direct argument of $::$, and so Coq rejects this productive definition.

As indicated by Coquand [5], the problem of guardedness in (1), the definition of nats, can be overcome by the alternative definition

$$\mathsf{nats} = \mathsf{nats\_from}\ 0 \qquad\qquad \mathsf{nats\_from}\ n = n :: \mathsf{nats\_from}\ (n+1)$$

The 'computation' is now embedded in the argument of the corecursion, and no longer obstructs the guarding constructor $::$. In Section 2 we give a similar solution for (2), by generalizing the corecursive construction to carry a nonempty list as argument. We show that every morphic sequence can be defined in Coq.

The main objective of this paper is to enable the use of circular coinductive rewriting [12] in Coq. In type theories, where proofs are first-class citizens, the problem of guardedness also occurs in proving coinductive statements. In particular, equational reasoning and rewriting on terms of a coinductive type may very well destroy guardedness. Let us consider an example where we want to show that two stream terms are bisimilar. Bisimilarity as a relation between streams can be defined coinductively as follows (where head is defined by $\mathsf{head}\ (x::s) = x$):

$$\frac{\mathsf{head}\ s = \mathsf{head}\ t \qquad \mathsf{tail}\ s \sim \mathsf{tail}\ t}{s \sim t}\ \sim_{\mathsf{intro}}$$

This means that $\sim$ is the greatest bisimulation (a bisimulation is a relation R such that for all stream terms related by R the heads are equal and the tails are again related by R). Let s and t be closed stream terms (i.e., containing no variables, only constants from a given signature). A proof of $s \sim t$ evaluates, in the limit, to an infinite constructor normal form $\sim_{\mathsf{intro}}\ d_0\ (\sim_{\mathsf{intro}}\ d_1\ (\sim_{\mathsf{intro}}\ d_2\ \cdots))$ where d_i is a proof of the equality of the i-th elements of s and of t, that is, $d_i : \mathsf{head}\ (\mathsf{tail}^i\ s) = \mathsf{head}\ (\mathsf{tail}^i\ t)$ for all $i \in \mathbb{N}$.[3]

Suppose we want to prove that alt is bisimilar to g alt, given the assumptions[4]:

$$\mathsf{alt} \sim 0 :: 1 :: \mathsf{alt} \qquad \mathsf{g}\ (0 :: s) \sim 0 :: 1 :: \mathsf{g}\ s \qquad \mathsf{g}\ (1 :: s) \sim \mathsf{g}\ s \tag{3}$$

[3] Throughout the paper, we use $=$ to denote Coq's equality, defined as the $\subseteq$-least reflexive relation, equivalent to Leibniz equality. We note that $=$ includes convertibility induced by Coq's native evaluation.

[4] The flexibility to use assumptions, in addition to definitions, is essential to allow for application of lemmas, and for unguarded or partial specifications of objects and functions. E.g., the second equation for g is not guarded. In fact, g is productive only for streams that contain infinitely many 0s.

Our coinductive proof that alt is a fixed point of g has the following shape:

$$\dfrac{\dfrac{d_0 \qquad \dfrac{d_1 \qquad d'' : \mathsf{tail}^2\ \mathsf{alt} \sim \mathsf{tail}^2\ (\mathsf{g}\ \mathsf{alt})}{\mathsf{tail}\ \mathsf{alt} \sim \mathsf{tail}\ (\mathsf{g}\ \mathsf{alt})}\ \sim_{\mathsf{intro}}}{\mathsf{alt} \sim \mathsf{g}\ \mathsf{alt}}\ \sim_{\mathsf{intro}}}{\mathsf{alt} \sim \mathsf{g}\ \mathsf{alt}}\ \mathsf{cofix}\ \pi \tag{d}$$

We explain the proof tree in a bottom-up fashion. By the rule cofix π we first introduce the coinduction hypothesis π : alt $\sim$ g alt into the context.[5] Next we apply the constructor $\sim_{\mathsf{intro}}$ twice. Here we have not displayed proofs d_0 and d_1 of type head alt = head (g alt) and head (tail alt) = head (tail (g alt)), respectively; both are obtained by plain equational reasoning and cause no problem. The point we want to make concerns the subproof d'' of tail^2 alt $\sim$ tail^2 (g alt). Left- and right-hand side can be converted by rewriting, using the hypothesis π and the assumptions (3), as follows:

$$\begin{aligned}&\mathsf{tail}^2\ \mathsf{alt} \xrightarrow{\mathsf{alt}} \mathsf{tail}^2\ (0 :: 1 :: \mathsf{alt}) \xrightarrow{\mathsf{tail}} \cdot \xrightarrow{\mathsf{tail}} \mathsf{alt} \xrightarrow{\pi} \mathsf{g}\ \mathsf{alt}\\ &\quad \xleftarrow{\mathsf{tail}} \cdot \xleftarrow{\mathsf{tail}} \mathsf{tail}^2\ (0 :: 1 :: (\mathsf{g}\ \mathsf{alt})) \xleftarrow{\mathsf{g}} \cdot \xleftarrow{\mathsf{g}} \mathsf{tail}^2\ (\mathsf{g}\ (0 :: 1 :: \mathsf{alt})) \xleftarrow{\mathsf{alt}} \mathsf{tail}^2\ (\mathsf{g}\ \mathsf{alt})\end{aligned}$$

This conversion gives rise to the proof tree d'' of the form:

$$\dfrac{\dfrac{e_1 : \mathsf{tail}^2\ \mathsf{alt} \sim \mathsf{alt} \qquad \dfrac{\pi : \mathsf{alt} \sim \mathsf{g}\ \mathsf{alt} \qquad e_2 : \mathsf{g}\ \mathsf{alt} \sim \mathsf{tail}^2\ (\mathsf{g}\ \mathsf{alt})}{\mathsf{alt} \sim \mathsf{tail}^2\ (\mathsf{g}\ \mathsf{alt})}\ \sim_{\mathsf{trans}}}{\mathsf{tail}^2\ \mathsf{alt} \sim \mathsf{tail}^2\ (\mathsf{g}\ \mathsf{alt})}\ \sim_{\mathsf{trans}} \tag{d''}$$

The omitted proofs e_1 and e_2 are obtained by equational reasoning without the use of the coinduction hypothesis π. But let us reconsider the proof tree d with d'' filled in by the concrete subtree above. Note that the corecursive call (or coinduction hypothesis) π is not a direct argument of the guarding constructor $\sim_{\mathsf{intro}}$, but nested within applications of $\sim_{\mathsf{trans}}$, as becomes even more apparent in the corresponding proof term[6]:

$$d = \mathsf{cofix}\ \pi\ (\sim_{\mathsf{intro}}\ d_0\ (\sim_{\mathsf{intro}}\ d_1\ (\sim_{\mathsf{trans}}\ e_1\ (\sim_{\mathsf{trans}}\ \pi\ e_2)))) \tag{4}$$

This corecursive construction is not guarded and Coq rejects it. However the term d is productive, and reduces to a constructor normal form in the limit.

Guardedness problems occur in the vast majority of cases when coinduction is combined with equational reasoning. The transformation of (productive) proof terms into guarded proof terms is the main topic of our paper. For this purpose we adopt techniques from process algebra and employ the method of 'bisimulation-up-to' [18], a generalization of Milner's bisimulation up to bisimilarity [16]. Let R be a binary relation on streams, and $\mathcal{U}$ a function from relations to relations. Then R is a bisimulation up to $\mathcal{U}$ if $\langle s, t\rangle \in R$ implies head s = head t and $\langle \mathsf{tail}\ s, \mathsf{tail}\ t\rangle \in \mathcal{U}(R)$. Under certain conditions, the fact that a relation R is a bisimulation up to $\mathcal{U}$ is sufficient to conclude that R is a subrelation of $\sim$, and

[5] Of course, now concluding the proof by π immediately yields a non-productive term, and is rightfully rejected by the guardedness checker.

[6] In a form analogous to stream definitions (1) and (2), the bisimilarity proof term (4) can also be written as $d = \sim_{\mathsf{intro}}\ d_0\ (\sim_{\mathsf{intro}}\ d_1\ (\sim_{\mathsf{trans}}\ e_1\ (\sim_{\mathsf{trans}}\ d\ e_2)))$.

so stream terms related by R are bisimilar. Typically R is included in $\mathcal{U}(R)$, and thus, in comparison with a full bisimulation, less diagrams have to be checked.

For the purpose of formalizing circular coinduction in Coq, we take $\mathcal{U}(R)$ to be the least relation including R and $\sim$, and closed under causal functions, transitivity and symmetry. A stream function F is *causal* (called 'special' in [15]) if the first n elements of the resulting stream $\mathsf{F}\ s$ only depend on the first n elements of the argument stream s. For the validity of a bisimilarity proof, it does not harm to use the coinduction hypothesis under a causal function [15].

We show soundness of the bisimulation-up-to method for this mapping $\mathcal{U}$, that is, if R is a bisimulation up to $\mathcal{U}$, then $\mathcal{U}(R)$ is a bisimulation. We also show that every circular coinduction proof can be transformed, in a structure-preserving way, to a proof that R is a bisimulation up to $\mathcal{U}$, where the relation R consists of all pairs (u, v) such that $u \sim v$ is a coinduction hypothesis in the original proof.

We thereby overcome the guardedness problem. The reason is that for proving that a relation is a bisimulation-up-to there is no need for corecursion, and hence guardedness is not an issue. The corecursive construction of bisimilarity proofs is now part of the general soundness result. In order to formalize a proof by circular coinduction in Coq, one can use our translation to obtain a proof by bisimulation-up-to $\mathcal{U}$, and then apply the soundness result to obtain a (guarded) bisimilarity proof accepted by Coq.

Related Work. Danielsson [6] works around the guardedness problem for stream definitions by defining a problem-specific language where the functions that obstruct guardedness are constructors, and defining an interpreter for the language by guarded corecursion. Recent work [14] supports compositionality in coinduction proofs by what is called 'parameterized coinduction', which allows for semantic rather than syntactic guardedness checking. In the present paper we are concerned with equational reasoning, and provide a systematic way for a Coq formalization of proofs by circular coinduction.

Overview. Sections 2 and 3 form a step-up to the main topic treated in Section 4. This order chronologically reflects how we came about to use up-to techniques. Sections 2 and 3 provide an informal discussion of how to overcome guardedness problems for several examples. In particular, in Section 2 we show how to define morphic sequences in Coq, using the idea explained above: postpone computation (viz. iterations of the morphism) in favour of guarding the corecursive call. The same idea is used in Section 3: to ensure guardedness of bisimilarity proofs, applications of transitivity are postponed by reformulating a goal $s \sim t$ into the equivalent statement $\forall s', t'.\ s' \sim s \Rightarrow t \sim t' \Rightarrow s' \sim t'$. In Section 4 we give a proof system for circular coinduction, restricted to the setting of streams over a two-element alphabet. We define our notion of bisimulation up-to $\mathcal{U}$, and discuss the soundness proof which states that if R is a bisimulation-up-to $\mathcal{U}$ then $\mathcal{U}(R)$ is a bisimulation. Finally we show that every proof by circular coinduction can be transformed to a bisimilarity proof accepted by Coq. The supporting Coq development is available as [10]. We conclude in Section 6.

2 Morphic Sequences in Coq

We show how to define morphic sequences by means of guarded corecursion. A morphic sequence (typically) is an infinite sequence obtained as the iterative fixed point of a morphism (also called a 'substitution'). Morphisms for transforming and generating infinite words provide a fundamental tool for formal languages, and have been studied extensively; we refer to [3]. We first give a standard definition of morphic sequences. Then we provide a 'direct' corecursive definition, using a productive version of the fixed point equation $h(w) = w$. Finally we show how to turn such an equation into a definition by guarded corecursion, and prove that the thus defined sequence is indeed the unique fixed point of h.

A *morphism* is a map $h : A^* \to B^*$, with A and B finite alphabets, such that $h(\varepsilon) = \varepsilon$ and $h(uv) = h(u)h(v)$ for all words $u, v \in A^*$, and can thus be defined by giving its values on the symbols of A.[7]

Let $h : A^* \to A^*$ be a morphism *prolongable* on the letter $a_0 \in A$, that is, $h(a_0) = a_0x$ for some $x \in A^*$ such that $h^i(x) \neq \varepsilon$ for all $i \geq 0$. Then we see that $h^{i+1}(a_0) = h^i(h(a_0)) = h^i(a_0x) = h^i(a_0)h^i(x)$, and hence $h^i(a_0)$ is a strict prefix of $h^{i+1}(a_0)$, for all $i \geq 0$. So then $\lim_{i\to\infty} h^i(a_0)$ exists and is infinite; this limit is denoted by $h^\omega(a_0) = \lim_{i\to\infty} h^i(a_0) = a_0\, x\, h(x) h^2(x)\, h^3(x) \cdots$ and it is readily seen to be the unique fixed point of h that starts with the letter a_0, that is, $h(h^\omega(a_0)) = h^\omega(a_0)$. A sequence $w \in A^\omega$ is *(purely) morphic* if $w = h^\omega(a_0)$ for some morphism h and starting letter a_0.

Without loss of generality [3], we may assume morphisms to be non-erasing, i.e., $h(a) \neq \varepsilon$ for all $a \in A$. Hence, we replace the condition that h be prolongable on starting letter a_0, by the simpler $h(a_0) = a_0x$ for some non-empty word x. Now we can define h as a function in $A^\omega \to A^\omega$ by guarded corecursion and pattern matching, as follows; for all $b \in A$ and $u \in A^\omega$:

$$h(b :: u) = b_0 :: b_1 :: \ldots :: b_{k-1} :: h(u) \qquad \text{where } b_0b_1 \cdots b_{k-1} = h(b),$$

We now give a productive (yet unguarded) definition of $w = h^\omega(a)$. This method is based on the work [7,8]. Clearly, the fixed point equation $w = h(w)$, where we now view w as a recursion variable, is not productive (and, typically, also does not have a unique solution for w). By using the knowledge that the first letter of w is a_0, we can turn it into

$$w = a_0 :: w' \qquad\qquad w' = \mathit{tail}(h(w)) \tag{5}$$

In order to see that this specification is productive indeed, we plug in the information that $h(a_0) = a_0x$ where, say, $x = a_1a_2 \cdots a_k$ with $k \geq 1$. We do so by replacing w by $a_0 :: w'$, in the right-hand side of the equation for w' and rewriting h and *tail*; we then obtain

$$w = a_0 :: w' \qquad\qquad w' = a_1 :: a_2 :: \ldots :: a_k :: h(w') \tag{6}$$

This is clearly a productive equation, because h 'consumes' one stream element at most, and, being non-erasing, produces one stream element at least. However, as will be clear by now, the corecursive equation for w' is not guarded, because

[7] Juxtaposition of words denotes concatenation.

the recursive call is nested within h.

In order to solve this problem, we generalize the construction by adding an argument on which h is applied, and from which we can always extract the next element to produce. The construction is reminiscent of Emile Post's tag systems [17]. Let the type of our morphism h be $A^+ \to A^+$, mapping nonempty words to nonempty words. Then we can define the morphic stream $w = h^\omega(a_0)$ by

$$w = a_0 :: tag_h(x) \qquad\qquad tag_h(ay) = a :: tag_h(yh(a)) \qquad (7)$$

where we recall that the morphism h, the starting letter a_0 and the non-empty word x are such that $h(a_0) = a_0x$. We note that the function $tag_h : A^+ \to A^\omega$ is defined by guarded corecursion, and hence (7) is accepted by Coq.

In this general set-up we prove (in Coq, see [10]) that the stream w defined by (7) is indeed the fixed point of h (and so satisfies also the equations (5) and (6)). That w is the unique fixed point (starting with a_0) follows from the fact that h is non-erasing and prolonging on a_0.

Example 1. The Fibonacci word $0100101001001\cdots$ [3] is generated by iterating the morphism f defined by $f(0) = 01$ and $f(1) = 0$, on the starting letter 0. In Coq we define nonempty lists inductively, we use $[a]$ for the singleton list, and overload the symbol :: to also denote the constructor for nonempty lists. The Fibonacci word fib is thus defined by

$$\begin{array}{lll}
\mathsf{fib} = 0 :: \mathsf{tag_f}\ [1] & \mathsf{tag_f}\ [a] = a :: \mathsf{tag_f}\ (\mathsf{f}\ a) & \mathsf{tag_f}\ (a :: u) = a :: \mathsf{tag_f}\ (u +\!\!+ \mathsf{f}\ a) \\
\mathsf{f}\ [0] = 0 :: [1] & \mathsf{f}\ (0 :: u) = 0 :: 1 :: \mathsf{f}\ u & [a] +\!\!+ v = a :: v \\
\mathsf{f}\ [1] = [0] & \mathsf{f}\ (1 :: u) = 0 :: \mathsf{f}\ u & (a :: u) +\!\!+ v = x :: (u +\!\!+ v)
\end{array}$$

3 Coinduction Loading

The idea of guarding the corecursive call by hiding the computation in an extra argument, as outlined in the previous section, turns out to be useful in the setting of bisimilarity proofs as well. We present the method of *coinduction loading*, which in some cases suffices to turn a productive bisimilarity proof into a guarded one. Here we want to avoid that transitivity of $\sim$ is applied to the coinduction hypothesis (= corecursive call). The idea is to reformulate a goal $s \sim t$ into the equivalent statement

$$\forall s', t'.\ s' \sim s \Rightarrow t \sim t' \Rightarrow s' \sim t'\,.$$

This enables us to move applications of transitivity to the argument of the corecursive call, as illustrated by the following example. Suppose we are given the following definitions (=) and assumption ($\sim$):

$$\begin{array}{ll}
\mathsf{dup}\ (x :: s) = x :: x :: \mathsf{dup}\ s & \mathsf{exp}\ (x :: s) \sim x :: \mathsf{dup}\ (\mathsf{exp}\ s) \\
\mathsf{odd}\ (x :: y :: s) = y :: \mathsf{odd}\ s & \mathsf{log}\ (x :: s) = x :: \mathsf{log}\ (\mathsf{odd}\ s)
\end{array}$$

The behavior of these functions is illustrated by applying them to the stream $\mathsf{nats} = 0 :: 1 :: 2 :: \ldots$:

$$\mathsf{dup}\ \mathsf{nats} = 0 :: 0 :: 1 :: 1 :: 2 :: 2 :: \ldots \qquad \mathsf{exp}\ \mathsf{nats} = 0 :: 1 :: 1 :: 2 :: 2 :: 2 :: 2 :: \ldots$$
$$\mathsf{odd}\ \mathsf{nats} = 1 :: 3 :: 5 :: 7 :: 9 :: \ldots \qquad \mathsf{log}\ \mathsf{nats} = 0 :: 2 :: 6 :: 14 :: 30 :: 62 :: \ldots$$

The assumption for exp can, in Coq, not be taken as a definition, because the corecursive call is nested within dup, and so is not guarded, although certainly productive. On the other hand, the corecursive equation for log is perfectly guarded. The function log has a logarithmically increasing production function [7], i.e., n elements in leads to $\lfloor \log_2(n+1) \rfloor$ elements out. Definitions are always preferable to assumptions since Coq's native evaluation does not harm guardedness of proofs, whereas rewriting terms by using bisimilarity proofs does. Consider the following corecursive proof term[8], which witnesses that odd ∘ dup is the identity function:

$$\mathsf{cofix}\ \pi\ (\lambda(x :: s').\ \sim_{\mathsf{intro}}\ (=_{\mathsf{refl}}\ x)\ (\pi\ s'))\ :\ \forall s.\ \mathsf{odd}\ (\mathsf{dup}\ s) \sim s \tag{8}$$

Note that this proof term is defined by guarded corecursion. After we have destructed s into $x :: s'$, the terms tail (odd (dup $(x :: s')$)) and odd (dup s') are convertible by Coq's native evaluation; similarly the subgoal head (odd (dup $(x :: s')$)) $= x$ is proved by reflexivity. Finally, the application of the coinduction hypothesis $\pi : \forall s.\ \mathsf{odd}\ (\mathsf{dup}\ s) \sim s$ to s' proves that odd (dup s') is bisimilar to s'.

Now we want to prove that also the composition log ∘ exp is the identity:

$$\forall s.\ \mathsf{log}\ (\mathsf{exp}\ s) \sim s \tag{9}$$

The proof that we want to construct (but which is not accepted by Coq) looks as follows; here we have omitted the subterms d of type head (log (exp $(x :: s')$)) $= x$, and e whose type is indicated in the tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{e : \mathsf{tail}\ (\mathsf{log}\ (\mathsf{exp}\ (x :: s'))) \sim \mathsf{log}\ (\mathsf{exp}\ s') \qquad \pi\ s' : \mathsf{log}\ (\mathsf{exp}\ s') \sim s'}{\mathsf{tail}\ (\mathsf{log}\ (\mathsf{exp}\ (x :: s'))) \sim s'}\ \sim_{\mathsf{trans}} \qquad d}{\mathsf{log}\ (\mathsf{exp}\ (x :: s')) \sim x :: s'}\ \sim_{\mathsf{intro}}}{\forall s.\ \mathsf{log}\ (\mathsf{exp}\ s) \sim s}\ \lambda(x :: s')}{\forall s.\ \mathsf{log}\ (\mathsf{exp}\ s) \sim s}\ \mathsf{cofix}\ \pi$$

Guardedness of the corecursive call $\pi\ s'$ is here obstructed by the application of $\sim_{\mathsf{trans}}$, transitivity of $\sim$. We explain why we cannot do without $\sim_{\mathsf{trans}}$ in this case. In order to prove that tail (log (exp $(x :: s')$)) is bisimilar to s' we cannot use Coq's native evaluation, for example because exp is not a defined function. Instead we proceed by rewriting using the coinduction hypothesis $\pi\ s'$, the assumption for exp and the lemma (8), as follows:

$$\mathsf{tail}\ (\mathsf{log}\ (\mathsf{exp}\ (x :: s'))) \overset{\mathsf{exp}}{\to} \mathsf{tail}\ (\mathsf{log}\ (x :: \mathsf{dup}\ (\mathsf{exp}\ s'))) \overset{\mathsf{log}}{\to}$$
$$\mathsf{tail}\ (x :: \mathsf{log}\ (\mathsf{odd}\ (\mathsf{dup}\ (\mathsf{exp}\ s')))) \overset{\mathsf{tail}}{\to} \mathsf{log}\ (\mathsf{odd}\ (\mathsf{dup}\ (\mathsf{exp}\ s'))) \overset{(8)}{\to} \mathsf{log}\ (\mathsf{exp}\ s') \overset{\pi}{\to} s'$$

All these rewrite steps are connected by applications of $\sim_{\mathsf{trans}}$. The proof tree above arises from splitting this rewrite sequence at the term log (exp s') (the middle term in the displayed application of $\sim_{\mathsf{trans}}$). In whatever way we split the sequence, the coinduction hypothesis is argument of at least one application

[8] We use the syntax $\lambda(x :: s').\ t$ to denote abstraction and pattern matching at once.

of $\sim_{\mathsf{trans}}$, resulting in an unguarded proof term.

This can be fixed with coinduction loading, thereby turning the above proof of (9) into a guarded proof of the equivalent statement $\forall s, t_1, t_2.(t_1 \sim \mathsf{log}\ (\mathsf{exp}\ s)\ \Rightarrow\ s \sim t_2\ \Rightarrow\ t_1 \sim t_2)$, as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{d' \qquad (\pi\ s'\ (\mathsf{tail}\ t_1)\ (\mathsf{tail}\ t_2)\ d_1\ d_2) : \mathsf{tail}\ t_1 \sim \mathsf{tail}\ t_2}{t_1 \sim t_2}\ \sim_{\mathsf{intro}}}{t_1 \sim \mathsf{log}\ (\mathsf{exp}\ (x :: s')) \ \Rightarrow\ x :: s' \sim t_2 \ \Rightarrow\ t_1 \sim t_2}\ \lambda\gamma_1, \gamma_2}{\forall s, t_1, t_2.\ (t_1 \sim \mathsf{log}\ (\mathsf{exp}\ s) \ \Rightarrow\ s \sim t_2 \ \Rightarrow\ t_1 \sim t_2)}\ \lambda(x :: s'), t_1, t_2}{\forall s, t_1, t_2.\ (t_1 \sim \mathsf{log}\ (\mathsf{exp}\ s) \ \Rightarrow\ s \sim t_2 \ \Rightarrow\ t_1 \sim t_2)}\ \mathsf{cofix}\ \pi$$

We note that λ-abstractions (introductions of $\forall$ and $\Rightarrow$) do not obstruct guardedness. We ignore the term $d' : \mathsf{head}\ t_1 = \mathsf{head}\ t_2$ which is a modification of $d : \mathsf{head}\ (\mathsf{log}\ (\mathsf{exp}\ (x :: s'))) = x$ using the hypotheses $\gamma_1 : t_1 \sim \mathsf{log}\ (\mathsf{exp}\ (x :: s'))$ and $\gamma_2 : s \sim t_2$. The other subtrees, d_1 and d_2, are given by the trees below. Here $\mathsf{comp}_{\mathsf{tail}}$ is an instance of $\mathsf{comp}_{\mathsf{f}}$ referring to the compatibility of a unary stream function f, i.e., $\mathsf{f}\ s \sim \mathsf{f}\ t$ whenever $s \sim t$. In Coq, compatibility cannot be proved for arbitrary f, but for concrete instances this forms no problem.

$$\dfrac{\dfrac{\gamma_1 : t_1 \sim \mathsf{log}\ (\mathsf{exp}\ (x :: s'))}{\mathsf{tail}\ t_1 \sim \mathsf{tail}\ (\mathsf{log}\ (\mathsf{exp}\ (x :: s')))}\ \mathsf{comp}_{\mathsf{tail}} \qquad e}{d_1 : \mathsf{tail}\ t_1 \sim \mathsf{log}\ (\mathsf{exp}\ s')}\ \sim_{\mathsf{trans}} \qquad \dfrac{\gamma_2 : x :: s' \sim t_2}{d_2 : s' \sim \mathsf{tail}\ t_2}\ \mathsf{comp}_{\mathsf{tail}}$$

Instead of giving a more formal definition of the transformation suggested by the above example, we continue our exposition with incorporating bisimulation-up-to techniques. The transformation described above will turn out to be an instance of the theory of bisimulation-up-to, namely as bisimulations up to transitivity and bisimilarity.

4 Circular Coinduction

We introduce a proof system for circular coinduction [12,15,19,21]. For the sake of presentation, we focus on streams over $\{0, 1\}$, but it is straightforward to generalize the method to infinite terms (ranked trees).

We assume that the signature is declared in Coq. We emphasize that the terms introduced below are just a notation for Coq terms. We have sorts $\mathfrak{B}$ and $\mathfrak{S}$ for $\{0, 1\}$ and streams over $\{0, 1\}$, respectively. A *signature* Σ is a set of symbols each having a fixed *type* in $\{\mathfrak{B}, \mathfrak{S}\}^* \times \{\mathfrak{B}, \mathfrak{S}\}$. We write $f : t_1 \times \ldots \times t_n \to s$ whenever $f \in \Sigma$ has type $\langle\langle t_1, \ldots, t_n\rangle, s\rangle$. Let Σ be a signature, and $\mathcal{X} = \mathcal{X}_{\mathfrak{B}} \cup \mathcal{X}_{\mathfrak{S}}$ be a set of variables such that $\mathcal{X} \cap \Sigma = \varnothing$. The set of *data terms* $T_{\mathfrak{B}}$ and *stream terms* $T_{\mathfrak{S}}$ over Σ and $\mathcal{X}$ are inductively defined by the grammar:

$$T_s ::= x \mid f(T_{s_1}, \ldots, T_{s_n}) \qquad (x \in \mathcal{X}_s,\ f \in \Sigma,\ f : s_1 \times \ldots \times s_n \to s)$$

for $s \in \{\mathfrak{B}, \mathfrak{S}\}$. We write T for $T_{\mathfrak{B}} \cup T_{\mathfrak{S}}$. A *substitution* is a mapping $\sigma : \mathcal{X} \to T$ that respects the sorts, that is, $\sigma(x) \in T_{\mathfrak{B}}$ for every $x \in \mathcal{X}_{\mathfrak{B}}$, and $\sigma(x) \in T_{\mathfrak{S}}$ for every $x \in \mathcal{X}_{\mathfrak{S}}$. We write $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ to abbreviate the substitution defined by $\sigma(x_1) = t_1$, $\ldots$, $\sigma(x_n) = t_n$ and $\sigma(x) = x$ for every $x \notin \{x_1, \ldots, x_n\}$.

For terms $s \in T$ and substitutions σ, we define s^σ inductively by $f(t_1, \ldots, t_n)^\sigma = f(t_1^\sigma, \ldots, t_n^\sigma)$ and $x^\sigma = \sigma(x)$. A *stream context* C is a term of sort $\mathfrak{S}$ over Σ and $\mathcal{X} \cup \{\Box\}$ where $\Box$ is a fresh variable of sort $\mathfrak{S}$. For terms $s \in T_{\mathfrak{S}}$ and contexts C, we write $C[s]$ for the term $C^{\Box \mapsto s}$.

We define *bisimilarity up to depth* n on stream terms inductively as follows:

$$\frac{}{s \sim_0 t} \qquad\qquad \frac{\mathsf{head}\ s = \mathsf{head}\ t \qquad \mathsf{tail}\ s \sim_n \mathsf{tail}\ t}{s \sim_{n+1} t}$$

A *causal context* is a context C such that for all stream terms s, t we have:

$$s \sim_n t \;\Rightarrow\; C[s] \sim_n C[t]\,, \quad \text{for all } n \in \mathbb{N}.$$

Examples of causal contexts are $\mathsf{dup}\ \Box$, $\mathsf{exp}\ \Box$, and $\mathsf{log}\ (\mathsf{exp}\ \Box)$; examples of non-causal contexts are $\mathsf{tail}\ \Box$, $\mathsf{odd}\ \Box$, and $\mathsf{log}\ \Box$, (see previous section). Note that for every causal context C we have that $\mathsf{tail}\ C[a :: \Box]$ is again causal.

Definition 2. The set Π of *(circular coinduction) proof terms* is inductively defined as follows (the superscript 'cc' stands for circular coinduction):

$$\begin{aligned}\Pi \;::=\; & \sim^{\mathsf{cc}}_{\mathsf{hyp}}\ \gamma\ C\ \sigma \mid \sim^{\mathsf{cc}}_{\mathsf{cut}}\ \gamma\ \Pi\ \Pi \mid \sim^{\mathsf{cc}}_{\mathsf{cohyp}}\ \delta\ D\ \sigma \mid \sim^{\mathsf{cc}}_{\mathsf{coin}}\ \delta\ E\ \Pi \mid \\ & \sim^{\mathsf{cc}}_{\mathsf{refl}}\ s \mid \sim^{\mathsf{cc}}_{\mathsf{sym}}\ \Pi \mid \sim^{\mathsf{cc}}_{\mathsf{trans}}\ \Pi\ \Pi \mid \sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{B}}}\ x\ \Pi\ \Pi \mid \sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{S}}}\ y\ \Pi\end{aligned}$$

where γ, δ are names for hypotheses, $s \in T_{\mathfrak{S}}$ is a stream term, $x \in \mathcal{X}_{\mathfrak{B}}$ and $y \in \mathcal{X}_{\mathfrak{S}}$ are variables of type $\mathfrak{B}$ and $\mathfrak{S}$, respectively, $\sigma : \mathcal{X} \to T$ is a substitution, and $C, D \in T$ are stream contexts, with D causal. The class E of equational proofs on data terms (equality of the heads) is left implicit.

The constructor $\sim^{\mathsf{cc}}_{\mathsf{coin}}$ represents the combination of the Coq constructs cofix and $\sim_{\mathsf{intro}}$ introduced earlier. We leave universal quantification implicit. That is, for stream terms s, t that contain variables $x_1, \ldots, x_n$, and for a relation R on stream terms, we write $s\ R\ t$ to denote $\forall x_1, \ldots, x_n.\ s\ R\ t$.

We now define 'typing judgments' for proof terms depending on two proof contexts Γ and Δ, consisting of triples written as $\gamma : s \sim^{\mathsf{cc}} t$. Here Γ contains assumptions, and Δ contains the coinduction hypotheses that are introduced in the construction of the proof. The intuition is that $\Gamma, \Delta \vdash d : s \sim^{\mathsf{cc}} t$ means that for all $n \in \mathbb{N}$, if all pairs in Γ are (fully) bisimilar, and all pairs in Δ are bisimilar up to depth n, then s is bisimilar to t up to depth $n+1$. The semantics for judgments, will be given in Section 5 where we translate them into Coq bisimilarity.

Definition 3. Let Γ, Δ be sets of triples $\gamma : u \sim^{\mathsf{cc}} v$, where every name γ appears at most once in $\Gamma \cup \Delta$. Let $d \in \Pi$ be a proof term, and s, t stream terms. We define the *judgment* $\Gamma, \Delta \vdash d : s \sim^{\mathsf{cc}} t$ inductively by the rules in Figure 1.

Next we introduce rewriting as a syntax for constructing proof terms.

Definition 4. Let $\gamma : u \sim^{\mathsf{cc}} v \in \Gamma$ and $\delta : u \sim^{\mathsf{cc}} v \in \Delta$. Let C be a context, σ a substitution, and abbreviate $s = C[u^\sigma]$ and $t = C[v^\sigma]$. Then we define

$$s \xrightarrow{\gamma} t = (\sim^{\mathsf{cc}}_{\mathsf{hyp}}\ \gamma\ C\ \sigma) \qquad\qquad s \xleftarrow{\gamma} t = (\sim^{\mathsf{cc}}_{\mathsf{sym}}\ (\sim^{\mathsf{cc}}_{\mathsf{hyp}}\ \gamma\ C\ \sigma))$$

$$\frac{}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{hyp}}\ \gamma\ C\ \sigma) : C[s^\sigma] \sim^{\mathsf{cc}} C[t^\sigma]}\ (\gamma : s \sim^{\mathsf{cc}} t) \in \Gamma$$

$$\frac{}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{cohyp}}\ \delta\ D\ \sigma) : D[s^\sigma] \sim^{\mathsf{cc}} D[t^\sigma]}\ (\delta : s \sim^{\mathsf{cc}} t) \in \Delta,\ D \text{ is causal}$$

$$\frac{}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{refl}}\ s) : s \sim^{\mathsf{cc}} s} \qquad \frac{\Gamma, \Delta \vdash d : t \sim^{\mathsf{cc}} s}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{sym}}\ d) : s \sim^{\mathsf{cc}} t}$$

$$\frac{\Gamma, \Delta \vdash d_1 : s \sim^{\mathsf{cc}} u \quad \Gamma, \Delta \vdash d_2 : u \sim^{\mathsf{cc}} t}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{trans}}\ d_1\ d_2\ ) : s \sim^{\mathsf{cc}} t}$$

$$\frac{\Gamma \vdash d_0 : (\mathsf{head}\ s = \mathsf{head}\ t) \quad \Gamma, \Delta \cup \{\delta : s \sim^{\mathsf{cc}} t\} \vdash d' : (\mathsf{tail}\ s \sim^{\mathsf{cc}} \mathsf{tail}\ t)}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{coin}}\ \delta\ d_0\ d') : s \sim^{\mathsf{cc}} t}\ \delta \notin \Delta$$

$$\frac{\Gamma, \Delta \vdash d_0 : (s \sim^{\mathsf{cc}} t)^{x \mapsto 0} \quad \Gamma, \Delta \vdash d_1 : (s \sim^{\mathsf{cc}} t)^{x \mapsto 1}}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{B}}}\ x\ d_0\ d_1) : s \sim^{\mathsf{cc}} t}\ x \in \mathcal{X}_{\mathfrak{B}}$$

$$\frac{\Gamma, \Delta \vdash d : (s \sim^{\mathsf{cc}} t)^{x \mapsto y :: z}}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{S}}}\ x\ d) : s \sim^{\mathsf{cc}} t}\ y \in \mathcal{X}_{\mathfrak{B}},\ z \in \mathcal{X}_{\mathfrak{S}} \text{ fresh for } s, t; \text{ and } x \in \mathcal{X}_{\mathfrak{S}}$$

$$\frac{\Gamma \cup \{\gamma : u \sim^{\mathsf{cc}} v\}, \Delta \vdash d_1 : s \sim^{\mathsf{cc}} t \quad \Gamma, \varnothing \vdash d_2 : u \sim^{\mathsf{cc}} v}{\Gamma, \Delta \vdash (\sim^{\mathsf{cc}}_{\mathsf{cut}}\ \gamma\ d_1\ d_2) : s \sim^{\mathsf{cc}} t}\ \gamma \notin \Gamma$$

Fig. 1. Proof rules for circular coinduction for stream terms.

$$s \xrightarrow{\delta} t = (\sim^{\mathsf{cc}}_{\mathsf{cohyp}}\ \delta\ C\ \sigma) \qquad\qquad s \xleftarrow{\delta} t = (\sim^{\mathsf{cc}}_{\mathsf{sym}}\ (\sim^{\mathsf{cc}}_{\mathsf{cohyp}}\ \delta\ C\ \sigma))$$

where in the case of $\xrightarrow{\delta}$ and $\xleftarrow{\delta}$, C is additionally required to be a causal context. Furthermore, let $\Xi = \{\xrightarrow{\gamma}, \xleftarrow{\gamma} \mid \gamma \in \Gamma\} \cup \{\xrightarrow{\delta}, \xleftarrow{\delta} \mid \delta \in \Delta\}$. Then for every $\leftrightarrow \in \Xi$ the judgment $\Gamma, \Delta \vdash s \leftrightarrow t : s \sim^{\mathsf{cc}} t$ holds. We define $s_0 \leftrightarrow_1 s_1 \leftrightarrow_2 \ldots \leftrightarrow_n s_n$ inductively by $(\sim^{\mathsf{cc}}_{\mathsf{trans}}\ (s_0 \leftrightarrow_1 s_1)\ (s_1 \leftrightarrow_2 \ldots \leftrightarrow_n s_n)\ )$ where $\leftrightarrow_i \in \Xi$ for $1 \leq i \leq n$.

For uniformity we work with assumptions only, but functions specified by guarded corecursion (like log in Section 3) can of course be taken as definitions in Coq, and then rewriting comes 'for free', i.e., are not reflected in the proof tree.

The following example illustrates the use of the syntax of Definition 4.

Example 5. Given the assumptions[9] $\Gamma = \{\gamma_1 : z_1 \sim^{\mathsf{cc}} (0 :: z_2), \gamma_2 : z_2 \sim^{\mathsf{cc}} (0 :: z_1)\}$, our goal is to construct a proof term witnessing $z_1 \sim^{\mathsf{cc}} z_2$. As usual we first apply the $\sim^{\mathsf{cc}}_{\mathsf{coin}}$-rule, i.e., we assume what we have to prove as a coinduction hypothesis $\delta : z_1 \sim^{\mathsf{cc}} z_2$, and then construct terms d_0 and d' so that

$$\frac{\Gamma \vdash d_0 : \mathsf{head}\ z_1 = \mathsf{head}\ z_2 \qquad \Gamma, \{\delta : z_1 \sim^{\mathsf{cc}} z_2\} \vdash d' : \mathsf{tail}\ z_1 \sim^{\mathsf{cc}} \mathsf{tail}\ z_2}{\Gamma, \varnothing \vdash (\sim^{\mathsf{cc}}_{\mathsf{coin}}\ \delta\ d_0\ d') : z_1 \sim^{\mathsf{cc}} z_2}\ \sim^{\mathsf{cc}}_{\mathsf{coin}}$$

[9] These mutual corecursive equations can actually be taken as definitions in Coq.

By rewriting we obtain

$$d_0 \;=\; \mathsf{head}\ z_1 \xrightarrow{\gamma_1} \mathsf{head}\ (0 :: z_2) \xrightarrow{\gamma_{\mathsf{head}}} 0 \xleftarrow{\gamma_{\mathsf{head}}} \mathsf{head}\ (0 :: z_1) \xleftarrow{\gamma_2} \mathsf{head}\ z_2$$

$$d' \;=\; \mathsf{tail}\ z_1 \xrightarrow{\gamma_1} \mathsf{tail}\ (0 :: z_2) \xrightarrow{\gamma_{\mathsf{tail}}} z_2 \xleftarrow{\delta} z_1 \xleftarrow{\gamma_{\mathsf{tail}}} \mathsf{tail}\ (0 :: z_1) \xleftarrow{\gamma_2} \mathsf{tail}\ z_2$$

So the proof tree corresponding to d' is as follows, where we write s' to denote $\mathsf{tail}\ s$, and where we omit contexts and proof terms:

$$\dfrac{\dfrac{\gamma_1 : z_1 \sim^{\mathsf{cc}} (0 :: z_2)}{z_1' \sim^{\mathsf{cc}} (0 :: z_2)'}\sim^{\mathsf{cc}}_{\mathsf{hyp}} \qquad \dfrac{\dfrac{\gamma_{\mathsf{tail}} : (x :: s)' \sim^{\mathsf{cc}} s}{(0 :: z_2)' \sim^{\mathsf{cc}} z_2}\sim^{\mathsf{cc}}_{\mathsf{hyp}} \qquad \dfrac{\dfrac{\dfrac{\delta : z_1 \sim^{\mathsf{cc}} z_2}{z_1 \sim^{\mathsf{cc}} z_2}\sim^{\mathsf{cc}}_{\mathsf{cohyp}}}{z_2 \sim^{\mathsf{cc}} z_1}\sim^{\mathsf{cc}}_{\mathsf{sym}} \qquad \dfrac{\dfrac{\gamma_{\mathsf{tail}} : (x :: s)' \sim^{\mathsf{cc}} s}{(0 :: z_1)' \sim^{\mathsf{cc}} z_1}\sim^{\mathsf{cc}}_{\mathsf{hyp}}}{z_1 \sim^{\mathsf{cc}} (0 :: z_1)'}\sim^{\mathsf{cc}}_{\mathsf{sym}} \qquad \dfrac{\dfrac{\gamma_2 : z_2 \sim^{\mathsf{cc}} 0 :: z_1}{z_2' \sim^{\mathsf{cc}} (0 :: z_1)'}\sim^{\mathsf{cc}}_{\mathsf{hyp}}}{(0 :: z_1)' \sim^{\mathsf{cc}} z_2'}\sim^{\mathsf{cc}}_{\mathsf{sym}}}{z_1 \sim^{\mathsf{cc}} z_2'}\sim^{\mathsf{cc}}_{\mathsf{trans}}}{z_2 \sim^{\mathsf{cc}} z_2'}\sim^{\mathsf{cc}}_{\mathsf{trans}}}{(0 :: z_2)' \sim^{\mathsf{cc}} z_2'}\sim^{\mathsf{cc}}_{\mathsf{trans}}}{z_1' \sim^{\mathsf{cc}} z_2'}$$

Next, we give a detailed example of a proof by circular coinduction, i.e., using the system introduced in Definition 3. In Section 5 we discuss how this proof is translated into a proof that Coq accepts. Consider the following guarded equations (here taken as assumptions) of functions $\mathsf{D}, \mathsf{T}, +$

$$\gamma_{\mathsf{D}} \;:\; \mathsf{D}\ s \sim^{\mathsf{cc}} s + \mathsf{tail}\ s \qquad\qquad \gamma_{\mathsf{T}} \;:\; \mathsf{T}\ s \sim^{\mathsf{cc}} \mathsf{head}\ s :: \mathsf{T}\ (\mathsf{D}\ s)$$

$$\gamma_{+} \;:\; (x :: s) + (y :: t) \sim^{\mathsf{cc}} (x + y) :: (s + t)$$

where $0 + 0 = 1 + 1 = 0$ and $0 + 1 = 1 + 0 = 1$.

Our goal is to prove by circular coinduction that T is an involution, that is, $\mathsf{T}\ (\mathsf{T}\ s) \sim s$, for all stream terms s. For this we use some easily proven facts about addition, and distribution of D over $+$:

$$\gamma_{+\mathsf{ass}} \;:\; x + (y + z) \sim^{\mathsf{cc}} (x + y) + z \qquad\qquad \gamma_{+\mathsf{com}} \;:\; x + y \sim^{\mathsf{cc}} y + x$$

$$\gamma_{+\mathsf{id}} \;:\; x + \mathsf{zeros} \sim^{\mathsf{cc}} x \qquad\qquad \gamma_{+\mathsf{ann}} \;:\; x + x \sim^{\mathsf{cc}} \mathsf{zeros}$$

$$\gamma_{\mathsf{Ddistr}} \;:\; \mathsf{D}\ (x + y) \sim^{\mathsf{cc}} \mathsf{D}\ x + \mathsf{D}\ y$$

and we let $\Gamma = \{\, \gamma_{\mathsf{head}}\,, \gamma_{\mathsf{tail}}\,, \gamma_{+}\,, \gamma_{\mathsf{D}}\,, \gamma_{\mathsf{T}}\,, \gamma_{+\mathsf{ass}}\,, \gamma_{+\mathsf{com}}\,, \gamma_{+\mathsf{id}}\,, \gamma_{+\mathsf{ann}}\,, \gamma_{\mathsf{Ddistr}} \,\}$.

Another lemma that we need, is distributivity of T over $+$. This in turn uses distributivity of D over $+$, which in Coq would destroy guardedness by invoking transitivity. The proof of $\mathsf{T}\ (x + y) \sim^{\mathsf{cc}} \mathsf{T}\ x + \mathsf{T}\ y$ has the following shape:

$$\dfrac{d_0 \qquad \dfrac{\dfrac{d' : \mathsf{tail}\ (\mathsf{T}\ ((a :: x') + (b :: y'))) \sim^{\mathsf{cc}} \mathsf{tail}\ (\mathsf{T}\ (a :: x') + \mathsf{T}\ (b :: y'))}{\mathsf{tail}\ (\mathsf{T}\ ((a :: x') + y)) \sim^{\mathsf{cc}} \mathsf{tail}\ (\mathsf{T}\ (a :: x') + \mathsf{T}\ y)}\sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{S}}}}{\mathsf{tail}\ (\mathsf{T}\ (x + y)) \sim^{\mathsf{cc}} \mathsf{tail}\ (\mathsf{T}\ x + \mathsf{T}\ y)}\sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{S}}}}{\mathsf{T}\ (x + y) \sim^{\mathsf{cc}} \mathsf{T}\ x + \mathsf{T}\ y}\sim^{\mathsf{cc}}_{\mathsf{coin}}\ \delta$$

We ignore the proof d_0 of $\mathsf{head}\ (\mathsf{T}\ (x + y)) \sim^{\mathsf{cc}} \mathsf{head}\ (\mathsf{T}\ x + \mathsf{T}\ y)$, but define d' by rewriting, using the coinduction hypothesis $\delta : \mathsf{T}\ (x + y) \sim^{\mathsf{cc}} \mathsf{T}\ x + \mathsf{T}\ y$:

$$\mathsf{tail}\ (\mathsf{T}\ ((a :: x') + (b :: y'))) \xrightarrow{\gamma_+}\xrightarrow{\gamma_{\mathsf{T}}}\xrightarrow{\gamma_{\mathsf{tail}}} \mathsf{T}\ (\mathsf{D}\ (x' + y')) \xrightarrow{\gamma_{\mathsf{Ddistr}}} \mathsf{T}\ (\mathsf{D}\ x' + \mathsf{D}\ y')$$
$$\xrightarrow{\delta} \mathsf{T}\ (\mathsf{D}\ x') + \mathsf{T}\ (\mathsf{D}\ y') \xleftarrow{\gamma_{\mathsf{tail}}}\xleftarrow{\gamma_+}\xleftarrow{\gamma_{\mathsf{T}}}\xleftarrow{\gamma_{\mathsf{T}}} \mathsf{tail}\ (\mathsf{T}\ (a :: x') + \mathsf{T}\ (b :: y'))$$

Thus we have shown that the following judgment holds using the proof system

for circular coinduction given in Figure 1:

$$\Gamma, \varnothing \vdash (\sim^{\mathsf{cc}}_{\mathsf{coin}}\ \delta\ (\sim^{\mathsf{cc}}_{\mathsf{case}_{\mathbb{S}}}\ x\ (\sim^{\mathsf{cc}}_{\mathsf{case}_{\mathbb{S}}}\ y\ d'))) : \mathsf{T}\ (x+y) \sim^{\mathsf{cc}} \mathsf{T}\ x + \mathsf{T}\ y$$

We now continue with showing that T is an involution. Let $\Gamma' = \Gamma \cup \{\gamma_{\mathsf{Tdistr}} : \mathsf{T}\ (x+y) \sim^{\mathsf{cc}} \mathsf{T}\ x + \mathsf{T}\ y\}$, i.e., we take up the lemma we have just proved as an assumption. The proof again starts by:

$$\frac{e_0 : \mathsf{head}\ (\mathsf{T}\ (\mathsf{T}\ s)) \sim^{\mathsf{cc}} \mathsf{head}\ s \qquad e' : \mathsf{tail}\ (\mathsf{T}\ (\mathsf{T}\ s)) \sim^{\mathsf{cc}} \mathsf{tail}\ s}{\mathsf{T}\ (\mathsf{T}\ s) \sim^{\mathsf{cc}} s} \sim^{\mathsf{cc}}_{\mathsf{coin}} \delta$$

and the proof term e' witnessing bisimilarity of $\mathsf{tail}\ (\mathsf{T}\ (\mathsf{T}\ s))$ and $\mathsf{tail}\ s$ is constructed by the following rewrite sequence:

$$\mathsf{tail}\ (\mathsf{T}\ (\mathsf{T}\ s)) \xrightarrow{\gamma_{\mathsf{T}}} \cdot \xrightarrow{\gamma_{\mathsf{tail}}} \mathsf{T}\ (\mathsf{D}\ (\mathsf{T}\ s)) \xrightarrow{\gamma_{\mathsf{D}}} \mathsf{T}\ (\mathsf{T}\ s + \mathsf{tail}\ (\mathsf{T}\ s)) \xrightarrow{\gamma_{\mathsf{T}}} \cdot \xrightarrow{\gamma_{\mathsf{tail}}} \mathsf{T}\ (\mathsf{T}\ s + \mathsf{T}\ (\mathsf{D}\ s))$$
$$\xrightarrow{\gamma_{\mathsf{Tdistr}}} \mathsf{T}\ (\mathsf{T}\ s) + \mathsf{T}\ (\mathsf{T}\ (\mathsf{D}\ s)) \xrightarrow{\delta} s + \mathsf{T}\ (\mathsf{T}\ (\mathsf{D}\ s)) \xrightarrow{\delta} s + \mathsf{D}\ s \xrightarrow{\gamma_{\mathsf{D}}} s + (s + \mathsf{tail}\ s)$$
$$\xrightarrow{\gamma_{+\mathsf{ass}}} (s+s) + \mathsf{tail}\ s \xrightarrow{\gamma_{+\mathsf{ann}}} \mathsf{zeros} + \mathsf{tail}\ s \xrightarrow{\gamma_{+\mathsf{id}}} \mathsf{tail}\ s$$

Here we have used the coinduction hypothesis δ twice. The first application (from left to right) under the causal context $\Box + \mathsf{T}\ (\mathsf{T}\ (\mathsf{D}\ s))$, and the second under the causal context $s + \Box$.

The above example illustrates several features of circular coinduction that cannot be captured by the method of coinduction loading introduced in the previous section (and certainly not by guarded corecursion). Without further generalizing, the method of coinduction loading cannot deal with more than one application of the coinduction hypothesis, and also does not allow for the use of the coinduction hypotheses under causal contexts.

The next section shows how to translate circular coinduction into Coq proofs.

5 Bisimulation-Up-To

To avoid the problems with guardedness in constructing a corecursive proof term for proving $s \sim t$, the user can instead define a relation R on stream terms with $\langle s, t\rangle \in R$, and then show that R is a bisimulation. This suffices to obtain a proof of $s \sim t$ in Coq, as follows: let $h : \forall s, t : A^\omega.\ s\ R\ t \Rightarrow \mathsf{head}\ s = \mathsf{head}\ t \wedge (\mathsf{tail}\ s)\ R\ (\mathsf{tail}\ t)$, witnessing that R is a bisimulation. A (Coq) proof term of type $\forall s, t.\ s\ R\ t \Rightarrow s \sim t$ is

$$\mathsf{cofix}\ \delta\ (\lambda s, t : A^\omega.\ \lambda\gamma : s\ R\ t.\ (\sim_{\mathsf{intro}}\ d_0\ (\delta\ (\mathsf{tail}\ s)\ (\mathsf{tail}\ t)\ d')))$$

where $d_0 = \mathsf{pj}_1\ (h\ s\ t\ \gamma)$ and $d' = \mathsf{pj}_2\ (h\ s\ t\ \gamma)$, with $\mathsf{pj}_i : p_1 \wedge p_2 \to p_i\ (i = 1, 2)$.

However, it is often cumbersome to construct such bisimulations. The reason is that for a relation R to be a bisimulation, it needs (a) to be closed under taking tail, (b) include all lemmas, and (c) all compositions of the required causal contexts. This typically gives rise to a large, or infinite relation.

We borrow a solution from process algebra [16,18], namely the method of *bisimulation-up-to*. A bisimulation-up-to is a relation which is included in a bisimulation, but which typically is not a bisimulation itself. A relation R is

a bisimulation-up-to $\mathcal{U}$ if for every $s\ R\ t$ we have $\mathsf{head}\ s = \mathsf{head}\ t$ and $\mathsf{tail}\ s\ \mathcal{U}(R)\ \mathsf{tail}\ t$. For suitable $\mathcal{U}$, it is possible to prove that $\mathcal{U}(R)$ is a bisimulation whenever R is a bisimulation-up-to $\mathcal{U}$. Since R can be substantially smaller than the enclosing bisimulation $\mathcal{U}(R)$, this may save a lot of work, because less pairs have to be checked.

Definition 6. Let R, R' be relations on stream terms. Then R *progresses to* R' if for every $s\ R\ t$ we have $\mathsf{head}\ s = \mathsf{head}\ t$ and $\mathsf{tail}\ s\ R'\ \mathsf{tail}\ t$.

Definition 7. Let $S, T \subseteq T_{\mathfrak{S}} \times T_{\mathfrak{S}}$, C a stream context and σ a substitution. We define $C[S] = \{\langle C[s], C[t]\rangle \mid \langle s,t\rangle \in S\}$, and $S^\sigma = \{\langle s^\sigma, t^\sigma\rangle \mid \langle s,t\rangle \in S\}$. For $x \in \mathcal{X}_{\mathfrak{B}}$ we let $S^{\mathsf{gen}_{\mathfrak{B}}(x)} = \{\langle s,t\rangle \mid \langle s^{x\mapsto i}, t^{x\mapsto i}\rangle \in S \text{ for all } i \in \{0,1\}\}$, and for $x \in \mathcal{X}_{\mathfrak{S}}$, $S^{\mathsf{gen}_{\mathfrak{S}}(x)} = \{\langle s,t\rangle \mid \langle s^{x\mapsto y::z}, t^{x\mapsto y::z}\rangle \in S \text{ for some } y \in \mathcal{X}_{\mathfrak{B}},\ z \in \mathcal{X}_{\mathfrak{S}} \text{ fresh for } s,t\}$. We abbreviate $S^{\mathsf{gen}} = \bigcup_{x\in\mathcal{X}_{\mathfrak{B}}} S^{\mathsf{gen}_{\mathfrak{B}}(x)} \cup \bigcup_{x\in\mathcal{X}_{\mathfrak{S}}} S^{\mathsf{gen}_{\mathfrak{S}}(x)}$. Also, we let $S^{-1} = \{\langle s,t\rangle \mid \langle t,s\rangle \in S\}$ and $S \cdot T = \{\langle s,t\rangle \mid \langle s,u\rangle \in S, \langle u,t\rangle \in T\}$.

Definition 8. Let $R \subseteq T_{\mathfrak{S}} \times T_{\mathfrak{S}}$. We define $\mathcal{U}(R)$ inductively by the grammar

$$\mathcal{U}(R) ::= R \mid \sim \mid \mathcal{U}(R)^\sigma \mid C[\mathcal{U}(R)] \mid \mathcal{U}(R)^{-1} \mid \mathcal{U}(R)\cdot\mathcal{U}(R) \mid \mathcal{U}(R)^{\mathsf{gen}}$$

where C is a causal context. R is a *bisimulation-up-to* $\mathcal{U}$ if R progresses to $\mathcal{U}(R)$.

In words, $\mathcal{U}(R)$ is the smallest relation that contains R and $\sim$, and is closed under substitution, causal contexts, symmetry, transitivity and generalization. Actually, the clauses for substitution and generalization are immediate in Coq, as there the pairs in $\mathcal{U}(R)$ are explicitly universally quantified.

Theorem 9 (Soundness). *If R is a bisimulation-up-to $\mathcal{U}$, then $\mathcal{U}(R)$ is a bisimulation.*

Proof. Let R be a bisimulation-up-to $\mathcal{U}$, and let s,t be terms such that $s\ \mathcal{U}(R)\ t$. We prove $\mathsf{head}\ s = \mathsf{head}\ t$ and $\mathsf{tail}\ s\ \mathcal{U}(R)\ \mathsf{tail}\ t$ by induction on the definition of $s\ \mathcal{U}(R)\ t$. We distinguish the following cases:

(i) $s\ R\ t$: follows from R being a bisimulation-up-to $\mathcal{U}$;

(ii) $s \sim t$: $\mathsf{head}\ s = \mathsf{head}\ t$ and $\mathsf{tail}\ s \sim \mathsf{tail}\ t$, and so $\mathsf{tail}\ s\ \mathcal{U}(R)\ \mathsf{tail}\ t$;

(iii) $s\ \mathcal{U}(R)^\sigma\ t$: for some $u, v \in T_{\mathfrak{S}}$, we have $s = u^\sigma$, $t = v^\sigma$ and $u\ \mathcal{U}(R)\ v$. By the induction hypothesis (IH) we have $\mathsf{head}\ u = \mathsf{head}\ v$ and so $\mathsf{head}\ s = \mathsf{head}\ t$; also $\mathsf{tail}\ u\ \mathcal{U}(R)\ \mathsf{tail}\ v$ by IH and so $\mathsf{tail}\ s\ \mathcal{U}(R)^\sigma\ \mathsf{tail}\ t$;

(iv) $s\ C[\mathcal{U}(R)]\ t$: for some $u, v \in T_{\mathfrak{S}}$, $s = C[u]$, $t = C[v]$, and $u\ \mathcal{U}(R)\ v$. Then $\mathsf{head}\ s = \mathsf{head}\ t$ follows from IH and causality of C; moreover, $\mathsf{tail}\ s\ D[\mathcal{U}(R)]\ \mathsf{tail}\ t$ follows from causality of $D = \mathsf{tail}\ C[\mathsf{head}\ u :: \Box]$ and IH;

(v) $s\ \mathcal{U}(R)^{-1}\ t$: direct from IH;

(vi) $s\ \mathcal{U}(R)\ u\ \mathcal{U}(R)\ t$: direct from IH;

(vii) $s\ \mathcal{U}(R)^{\mathsf{gen}_{\mathfrak{B}}(x)}\ t$: then $s^{x\mapsto i}\ \mathcal{U}(R)\ t^{x\mapsto i}$ for $i \in \{0,1\}$ and so by IH $\mathsf{head}\ s = \mathsf{head}\ t$ for all possible values of $x \in \mathcal{X}_{\mathfrak{B}}$, and $\mathsf{tail}\ s\ \mathcal{U}(R)^{\mathsf{gen}_{\mathfrak{B}}(x)}\ \mathsf{tail}\ s$ by IH;

(viii) $s\ \mathcal{U}(R)^{\mathsf{gen}_{\mathfrak{S}}(x)}\ t$: similar to previous case.

From a proof using circular coinduction, we extract a bisimulation-up-to $\mathcal{U}$:

Definition 10. Let $d : \Gamma, \Delta \vdash s \sim^{\mathsf{cc}} t$ be a proof using circular coinduction. We define R_d to consist of all pairs (u, v) such that the proof d contains a sub-proof of the form $\sim^{\mathsf{cc}}_{\mathsf{coin}} \ \ldots : u \sim^{\mathsf{cc}} v$.

In what follows, we assume all contexts to be compatible with bisimilarity, i.e., $u \sim v \ \Rightarrow \ C[u] \sim C[v]$. This is used in item (i) of the proof of Theorem 12 below. In practice, this assumption can be dropped as proving it for concrete C forms no problem.

Lemma 11. *If R progresses to $\overline{R}$, and S progresses to $\overline{S}$, then $R \cup S$ progresses to $\overline{R} \cup \overline{S}$.* □

Theorem 12. *Assume $\Gamma, \Delta \vdash d : s \sim^{\mathsf{cc}} t$ and $u \sim v$ for all pairs $\gamma : u \sim^{\mathsf{cc}} v$ in Γ. Let $R = R_d \cup \Delta$. Then $s \ \mathcal{U}(R)\ t$ and R_d progresses to $\mathcal{U}(R)$.*

Proof. The proof proceeds by induction on the structure of $\Gamma, \Delta \vdash d : s \sim^{\mathsf{cc}} t$ (see Figure 1), as follows. In each case, we let R abbreviate $R_d \cup \Delta$. In the first three cases, we have $R_d = \varnothing$ and so R_d trivially progresses to $\mathcal{U}(R)$.

(i) $\Gamma, \Delta \vdash d : C[u^\sigma] \sim^{\mathsf{cc}} C[v^\sigma]$ with $d = \sim^{\mathsf{cc}}_{\mathsf{hyp}} \ \gamma \ C \ \sigma$. Then $(\gamma : u \sim^{\mathsf{cc}} v) \in \Gamma$, and $u^\sigma \sim v^\sigma$ by assumption. Hence, by compatibility we have $C[u^\sigma] \sim C[v^\sigma]$ and, using $\sim \subseteq \mathcal{U}(R)$ we get $C[u^\sigma] \ \mathcal{U}(R)\ C[v^\sigma]$.

(ii) $\Gamma, \Delta \vdash d : D[u^\sigma] \sim^{\mathsf{cc}} D[v^\sigma]$ with $d = \sim^{\mathsf{cc}}_{\mathsf{cohyp}} \ \delta \ D \ \sigma$, and D a causal context. Then $(\delta : u \sim^{\mathsf{cc}} v) \in \Delta$ and so $u \ R\ v$, $u^\sigma \ \mathcal{U}(R)\ v^\sigma$ and $D[u^\sigma] \ \mathcal{U}(R)\ D[v^\sigma]$.

(iii) $\Gamma, \Delta \vdash d : u \sim^{\mathsf{cc}} u$ with $d = \sim^{\mathsf{cc}}_{\mathsf{refl}} u$. Then $u \ \mathcal{U}(R)\ u$ by reflexivity of $\sim$ and $\sim \subseteq \mathcal{U}(R)$.

(iv) $\Gamma, \Delta \vdash d : u \sim v$ with $d = \sim^{\mathsf{cc}}_{\mathsf{sym}} \ d_1$ and $\Gamma, \Delta \vdash d_1 : v \sim u$. Then $v \ \mathcal{U}(R)\ u$ by the induction hypothesis (IH), and $u \ \mathcal{U}(R)\ v$ by symmetry of $\mathcal{U}(R)$. Also, R_d progresses to $\mathcal{U}(R)$ by IH, because $R_d = R_{d_1}$.

(v) $\Gamma, \Delta \vdash d : u \sim^{\mathsf{cc}} v$ with $d = \sim^{\mathsf{cc}}_{\mathsf{trans}} \ d_1 \ d_2$, and $\Gamma, \Delta \vdash d_1 : u \sim^{\mathsf{cc}} w$ and $\Gamma, \Delta \vdash d_2 : w \sim^{\mathsf{cc}} v$ for some stream term w. We conclude $u \ \mathcal{U}(R)\ v$ from $u \ \mathcal{U}(R)\ w$, and $w \ \mathcal{U}(R)\ v$ by IH and transitivity of $\mathcal{U}(R)$. By IH we have that R_{d_1} progresses to $\mathcal{U}(R_{d_1} \cup \Delta)$, and R_{d_2} progresses to $\mathcal{U}(R_{d_2} \cup \Delta)$. By Lemma 11 $R_d = R_{d_1} \cup R_{d_2}$ progresses to $\mathcal{U}(R)$.

(vi) $\Gamma, \Delta \vdash d : u \sim^{\mathsf{cc}} v$ with $d = \sim^{\mathsf{cc}}_{\mathsf{coin}} \ \delta \ d_0 \ d'$ and $\Gamma \vdash d_0 : \mathsf{head}\ u = \mathsf{head}\ v$, and $\Gamma, \Delta' \vdash d' : \mathsf{tail}\ u \sim^{\mathsf{cc}} \mathsf{tail}\ v$ where $\Delta' = \Delta \cup \{\delta : u \sim^{\mathsf{cc}} v\}$. Note that $R_d = R_{d'} \cup \{\delta : u \sim^{\mathsf{cc}} v\}$ and so $R = R_d \cup \Delta = R_{d'} \cup \Delta'$. From the IH we obtain that $R_{d'}$ progresses to $\mathcal{U}(R_{d'} \cup \Delta') = \mathcal{U}(R)$. Moreover, $\{\delta : u \sim^{\mathsf{cc}} v\}$ progresses to $\mathcal{U}(R)$ since d_0 is a proof of $\mathsf{head}\ u = \mathsf{head}\ v$, and $\mathsf{tail}\ u \ \mathcal{U}(R)\ \mathsf{tail}\ v$ by IH. With Lemma 11 we conclude that R_d progresses to $\mathcal{U}(R)$. Furthermore, $u \ \mathcal{U}(R)\ v$ by $\langle u, v\rangle \in R_d \subseteq R \subseteq \mathcal{U}(R)$.

(vii) $\Gamma, \Delta \vdash d : u \sim^{\mathsf{cc}} v$ with $d = \sim^{\mathsf{cc}}_{\mathsf{cut}} \ \gamma \ d_1 \ d_2$ and $\Gamma', \Delta \vdash d_1 : u \sim^{\mathsf{cc}} v$, and $\Gamma, \varnothing \vdash d_2 : q \sim^{\mathsf{cc}} r$ for some stream terms q, r, where $\Gamma' = \Gamma \cup \{\gamma : q \sim^{\mathsf{cc}} r\}$. We note that $q \sim r$ holds by Theorem 9, since by IH we have $q \ \mathcal{U}(R_{d_2})\ r$, and R_{d_2}

progresses to $\mathcal{U}(R_{d_2})$. Hence $\Gamma' \subseteq \sim$, and by IH we obtain $u\ \mathcal{U}(R_{d_1} \cup \Delta)\ v$ and we conclude $u\ \mathcal{U}(R)\ v$ from $R_d = R_{d_1} \cup R_{d_2}$ (clearly, $\mathcal{U}$ is a monotone function with respect to $\subseteq$). Moreover, by IH R_{d_1} progresses to $\mathcal{U}(R_{d_1} \cup \Delta)$, and R_{d_2} progresses to $\mathcal{U}(R_{d_2})$. So by Lemma 11 we obtain that $R_d = R_{d_1} \cup R_{d_2}$ progresses to $\mathcal{U}(R)$.

(viii) $\Gamma, \Delta \vdash d : u \sim^{\mathsf{cc}} v$ with $d = \sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{B}}}\ x\ d_0\ d_1$ and $\Gamma, \Delta \vdash d_i : (u \sim^{\mathsf{cc}} v)^{x \mapsto i}$ for $i \in \{0, 1\}$ and $x \in \mathcal{X}_{\mathfrak{B}}$. By IH we get $u^{x \mapsto i}\ \mathcal{U}(R)\ v^{x \mapsto i}$ for $i \in \{0, 1\}$, hence $u\ \mathcal{U}(R)\ v$ by $\mathcal{U}(R)^{\mathsf{gen}_{\mathfrak{B}}} \subseteq \mathcal{U}(R)$. Furthermore, $R_d = R_{d_0} \cup R_{d_1}$ progresses to $\mathcal{U}(R)$ by IH and Lemma 11.

(ix) $\Gamma, \Delta \vdash d : u \sim^{\mathsf{cc}} v$ with $d = \sim^{\mathsf{cc}}_{\mathsf{case}_{\mathfrak{S}}}\ x\ e$ and $\Gamma, \Delta \vdash e : u^{x \mapsto y::z} \sim^{\mathsf{cc}} v^{x \mapsto y::z}$. From IH we obtain $u^{x \mapsto y::z}\ \mathcal{U}(R)\ v^{x \mapsto y::z}$. So $u\ \mathcal{U}(R)\ v$ follows from $\mathcal{U}(R)^{\mathsf{gen}_{\mathfrak{S}}} \subseteq \mathcal{U}(R)$. Furthermore, $R_d = R_e$ progresses to $\mathcal{U}(R)$ by IH.

From Theorems 9 and 12 it follows that every proof by circular coinduction can be transformed to a bisimilarity proof accepted by Coq. We have formalized Theorem 9 in Coq, see [10]. We are currently working on automating the translation of circular coinductive proofs as produced by the prover Circ [12,15], or Streambox [21] into Coq proofs.

Corollary 13. *If $\Gamma, \varnothing \vdash d : s \sim^{\mathsf{cc}} t$, then R_d is a bisimulation-up-to $\mathcal{U}$ and $s\ \mathcal{U}(R_d)\ t$. Hence $s \sim t$ is provable from Γ in Coq.* □

6 Discussion

Generally speaking, for *programming* with infinite objects, translating a productive specification into a guarded definition may take considerable effort. In doing so, the elegance and 'directness' of the original specification is often lost, thereby complicating further processing of the defined object. For programming, we therefore believe that alternative approaches are favorable. Here one may think of implementing a more flexible productivity checker by using a type-based approach as advocated in [13,1,20]. It would also be convenient if Coq would allow productivity of the corecursive program to be proved separately (for recursive programs, wellfoundedness can be proved separately in Coq). This then would open the door for more advanced tactics based on methods as described in [7,8,9].

However, for *reasoning* about infinite objects the situation is different, as we have shown. The reason is that for bisimilarity proofs the coinduction hypothesis is usually not subject to further pattern matching. This is in contrast to programming where recursive calls are usually manipulated further.

References

1. A. Abel. Termination and Productivity Checking with Continuous Types. In *Proc. 6th Conf. on Typed Lambda Calculi and Applications (TLCA 2003)*, volume 2701 of *LNCS*, pages 1–15. Springer, 2003.

2. The Agda team. *The Agda Wiki*, 2011. `http://wiki.portal.chalmers.se/agda`.
3. J.-P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, New York, 2003.
4. The Coq development team. *The Coq Proof Assistant Reference Manual*. LogiCal Project, 2012. `http://coq.inria.fr`, version 8.3.
5. Th. Coquand. Infinite Objects in Type Theory. In *Postproc. Workshop on Types for Proofs and Programs (TYPES 1993)*, volume 806 of *LNCS*, pages 62–78. Springer, 1994.
6. N. A. Danielsson. Beating the Productivity Checker Using Embedded Languages. In *Proc. Workshop on Partiality and Recursion in Interactive Theorem Provers (PAR 2010)*, volume 43 of *EPTCS*, pages 29–48, 2010.
7. J. Endrullis, C. Grabmayer, and D. Hendriks. Data-Oblivious Stream Productivity. In *Proc. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2008)*, volume 5330 of *LNCS*, pages 79–96. Springer, 2008.
8. J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J.W. Klop. Productivity of Stream Definitions. *Theoretical Computer Science*, 411, 2010.
9. J. Endrullis and D. Hendriks. Lazy Productivity via Termination. *Theoretical Computer Science*, 412(28):3203–3225, 2011.
10. J. Endrullis, D. Hendriks, and M. Bodin. Coq Formalization for Circular Coinduction, 2012. Available at `http://www.cs.vu.nl/~diem/research/up_to.tgz`.
11. E. Giménez. Codifying Guarded Definitions with Recursive Schemes. In *Postproc. Workshop on Types for Proofs and Programs (TYPES 1993)*, volume 806 of *LNCS*, pages 39–59. Springer, 1994.
12. J. Goguen, K. Lin, and G. Roşu. Circular Coinductive Rewriting. In *Proc. of Automated Software Engineering*, pages 123–131. IEEE, 2000.
13. J. Hughes, L. Pareto, and A. Sabry. Proving the Correctness of Reactive Systems Using Sized Types. In *Symposium on Principles of Programming Languages (POPL 1996)*, pages 410–423, 1996.
14. C.-K. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The Power of Parameterization in Coinductive Proof. In *Proc. Symp. on Principles of Programming Languages (POPL 2013)*, pages 193–206. ACM, 2013.
15. D. Lucanu and G. Roşu. Circular Coinduction with Special Contexts. In *Proc. Int. Conf. on Formal Methods and Software Engineering (ICFEM 2009)*, pages 639–659. Springer, 2009.
16. R. Milner. *Communication and Concurrency*. Prentice-Hall International Series in Computer Science. Prentice-Hall, 1989.
17. E. L. Post. Formal Reductions of the General Combinatorial Decision Problem. *American Journal of Mathematics*, (65):197–215, 1943.
18. D. Pous and D. Sangiorgi. Enhancements of the Coinductive Proof Method. In D. Sangiorgi and J. J. M. M. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2011. Chapter 6.
19. G. Roşu and D. Lucanu. Circular Coinduction: A Proof Theoretical Foundation. In *Proc. Conf. on Algebra and Coalgebra in Computer Science (CALCO 2009)*, volume 5728 of *LNCS*, pages 127–144. Springer, 2009.
20. P. Severi and F.-J. de Vries. Pure Type Systems with Corecursion on Streams: From Finite to Infinitary Normalisation. In *Proc. Int. Conf. on Functional Programming (ICFP 2012)*, pages 141–152. ACM, 2012.
21. H. Zantema and J. Endrullis. Proving Equality of Streams Automatically. In *Proc. Conf. on Rewriting Techniques and Applications (RTA 2011)*, volume 10 of *LIPIcs*, pages 393–408. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.